

METHOD AND SYSTEM FOR HARDWARE SIMULATION

FIELD OF THE INVENTION

[0001] This invention relates to methods and systems for simulation of hardware systems and in particular to simulation of packet network transport systems.

BACKGROUND OF THE INVENTION

[0002] During the design and development of embedded systems, engineers often attempt to meld hardware and software systems to perform a variety of tasks. However, in order to test a particular designed software or device embodying the software, the engineer must often incorporate the device or software into an existing system having many elements of hardware and software. Furthermore, testing can often involve changes to specific portions of hardware and software of the system in order to test the software capacity to function in a system having these changes. To provide such a testing system and to provide changes to various portions of the system can be expensive and difficult.

SUMMARY OF THE INVENTION

[0003] Accordingly, a testing environment is provided which allows software developers to test their code prior to the availability of hardware. This can provide great benefit to developers since hardware is expensive and the simulator can provide testing of various hardware systems. In addition to providing simulated hardware, non-destructive reading of memory registers is facilitated and testing of difficult to reproduce fault conditions such as queue

overflow is permitted. The simulator system also provides for automated testing of a suite of tests through a scripting interface.

BRIEF DESCRIPTION OF THE DRAWINGS

[0004] Fig. 1 is a block diagram of an overall architecture of an embodiment of a system for simulation of network transport in accordance with the invention;

[0005] Fig. 2 is a block diagram of an architecture for an embodiment of a software emulator in accordance with the invention in Figure 1;

[0006] Fig. 3 is a block diagram of an architecture for an embodiment of a network simulator in accordance with the invention in Figure 1;

[0007] Fig. 4 is a block diagram of linked objects for one embodiment of the hardware simulator in accordance with the invention in Figure 1;

[0008] Fig. 5 is a block diagram of a hierarchy of nodes for one embodiment of a hardware simulator in accordance with the invention in Figure 1;

[0009] Fig. 6 is a block diagram showing a relationship of node types for one embodiment of a hardware simulator in accordance with the invention in Figure 1; and

[0010] Fig. 7 is a block diagram of linked class objects for one embodiment of a hardware simulator in accordance with the invention in Figure 1.

[0011] Throughout the figures, the same reference numerals and characters, unless otherwise stated, are used to denote like features, elements, components or portions of the illustrated embodiments. Moreover, while the subject invention will now be described in detail with reference to the figures, it is done so in connection with the illustrative embodiments. It is intended that changes and modifications can be made to the described embodiments without departing from the true scope and spirit of the subject invention as defined by the appended claims.

2025-01-01 10:00:00

[0012]

DETAILED DESCRIPTION

[0013] The system for hardware simulation provides an environment in which embedded software developers can test their software code as implemented in a system without having to procure and install the actual hardware systems which form the environment in which the software is intended to operate.

[0014] Figure 1 shows an overall system architecture of one embodiment of the system for hardware simulation 100 according to the invention that provides a testing platform for embedded software. The system can be distributed among several computers or it can be integration within a single computer. However, to provide a realistic testing environment, at least two separate computer systems are used. A first embodiment is shown in Figure 1 in which an emulator system 101 and a simulation system 102 are provided as an example using two separate systems. At least one emulation system 101 is provided for emulating actions of the embedded software 103 which is to be tested.

[0015] The emulator system 101 includes a physical board having similar hardware to that which the embedded software 103 is intended to run on. The physical board is provided with a central processing unit (CPU) 110, on-board memory 111 and can have peripheral elements (not shown) such as Field Programmable Gate Arrays (FPGAs), communication ports which can include Ethernet and serial ports, among other things. One example of such an emulator system 101 can be a system board having a Receiver-Transmitter Pair (RTP) application specific integrated circuit (ASIC) mounted on the board. The RTP ASIC can have

embedded software for controlling the processes of the RTP and which is to be tested. The embedded software 101 can be executed in a real-time operating system (RTOS), such as pSOS, and run on the central processing unit (CPU) of the physical board.

[0016] One instance of the embedded software 103 is executed on each CPU of an emulator system 101. If the physical board is provided with more than one CPU, then a version of the embedded software 103 can be executed on each CPU. In addition, the system for hardware simulation 100 can provide for multiple physical board and emulator systems 101. When multiple emulator systems 101 are provided, each associated CPU can be given a unique address to direct or attribute communication to the appropriate emulator system 101 where Internet Protocol (IP) or similar communication is provided for.

[0017] Each emulator system 101 is provided with a Hardware Interface Layer (HIL) interface 104 through which all interactions with the hardware are passed and permit interactions with the real hardware to be replaced with interactions with the simulated hardware. The HIL is the lowest level software above the real hardware. Thus, the HIL interface provides the embedded software 103 with a means to communicate events in the emulator system 101 to the network simulator 105. Such communication is provided across at least one interface according to the specific communication protocol to be implemented by the embedded software 103. Read and write requests to target registers of devices are intercepted at the hardware interface layer. Accordingly, read and write calls are made at an interface at the hardware interface layer 202. The HIL interface 104 is integrated with the embedded software 103 to run on the same board and utilize the same resources -- such as CPU and associated memory -- of each emulator system 101.

[0018] The simulation system 102 is provided to carry out the details of simulating a system of components which interact with each emulator system 101. The simulation system 102 is provided with a network simulator 105, a simulation library 104, and can include end station simulators 109, a graphical user interface 108 and a scripting interface 107. The simulation system 102 can be executed on a single computer, such as a Solaris workstation, or it can be distributed among several computers.

[0019] The Graphical User Interface (GUI) 108 can be provided to give a user a graphical interface for controlling and viewing the states of the system. The GUI 108 provides for topology creation and allows a user to graphically compose the network topology using drag and drop mouse movements to specify hardware to be simulated and their interconnecting links with at least one emulator system 101. The GUI 108 can permit the user to specify which emulator system 101 instance is used for emulating a particular node in the topology of the network system being composed. The GUI 108 allows the user to create a network topology, start and stop the simulator, manipulate and inspect the memory map of any device in the network, monitor the packets on the links, and inject failures. Among other things, the GUI 108 can be programmed to perform the management functions on any simulated node and to program the end stations to perform specific tasks. The GUI 108 can provide a hardware generator and can be provided on one or more machines having separate functions. Greater detail for the operation of the GUI 108 is provided below.

[0020] The simulator library 106 is provided as an application programming interface to give a user a library of software function calls which can be requested through the GUI 108 or the scripting interface 107. Such processes can include creation of hardware components to be

simulated, programming of functionality of the end stations 109, reading diagnostic information of the simulated network, and introduction of errors into the system for testing among other things. In addition, the processes can be also be used to control the functionality implemented by the various components of the network simulator 105, among other things. Alternatively, the simulator library can be provided with an end station library for simulating a scripted end station 109.

[0021] The scripting interface 107 is provided to permit a user with a command line interface to request a particular function call of the simulator library. The scripting interface 107 can also process a script to request a sequence of operations to be carried out on the simulated network. The scripting interface 107 provides all of the function calls that the GUI 108 allows the user to perform. However, unlike the GUI 108, the scripting interface 107 can be used to write regression tests to be automated without user interaction. As a particular aspect of the embedded software 103 is modified, the scripting interface 107 can be used to automate a suite of tests of the embedded software 103 with the simulated network without requiring user interaction with the system during the tests.

[0022] At least one end station simulator 109 can be provided for simulating end stations which participate in the simulated network. End stations being simulated include such devices as subtending routers among other things. Each end station simulator 109 can be implemented either as a scripted or as a real interface. A scripted end station simulator 109 can be provided with limited functionality which only simulates those responses from the end stated simulator 109 which can be expected to be elicited by events from the emulator system 101. For example, a user can specify IP packet exchanges between an end station simulator 109 and its subtending

interface card in the network simulation by using a scripting interface language such as ITcl. The scripted end station simulator 109 is provided to communicate with the network simulator 105 through a TCP/IP connection. to an end station stub 307.

[0023] A real interface end station simulator 109 is provided to connect to a real end station – e.g., a router – over Ethernet or another medium, and provide that end station with communication with the simulated network. The real interface end station simulator 109 can receive any packet arriving on the real interface and deliver it to the end station stub 307 which then injects the packet into the simulated network as described by the user-specified links of the network topology. Communication between the real interface end station simulator 109 and the network simulator 105 can be provided via a TCP/IP connection.

[0024] One embodiment of the HIL interface 104 is shown in Figure 2. The HIL interface 104 includes an HIL stub 201 and a hardware interface layer 202. The hardware interface layer 202 is provided to filter communications from the embedded software 103 with the hardware on-board the emulator system 101. The hardware interface layer 202 includes a simulator stub 203 for intercepting memory calls to any simulated memory-mapped device. Simulated memory mapped devices include registers on peripheral devices or memory on simulated devices among other things. Read, and write requests from the embedded software 103 go through the hardware interface layer 202 and are intercepted by the simulator stub 201. Write calls are immediately returned and read calls are blocked until a response has been received. Memory calls from the embedded software 103 that are directed to on-board memory 111 of the emulator system 101 can be permitted by the simulator stub 203 to pass through. Messages that are intercepted by the simulator stub 203 are converted into a protocol suitable for inter-process

communication (IPC) and are sent to the HIL stub 201. The IPC message has a protocol, which at a low-level, is dependent upon the embedded operating system. At a higher layer, the messages can represent simulated events.

[0025] Depending upon the characteristics attributed to the simulated hardware, the HIL stub 201 can provide one or more communications controller 204 for communicating between the CPU 110 of the emulator system 101 and any peripherals associated with the emulator system 101. Communication controllers 204 can be provided to simulate behavior of simulated devices on the physical board and which are provided by the network simulator 105. For example, a simulated DMA controller can be provided for simulating transfer of data from a peripheral's memory to the physical board's main RAM. Since the peripheral's memory is simulated and the memory on the physical board is not, the communication controller 204 can give the simulator access to the physical board's RAM. To achieve this, a simulation of the DMA controller resides in the emulator system 101. Communication controllers 204 can also be used to process interrupt events from the simulated hardware of the emulator system 101. In such a case, the communication controller 204 receives interrupt events and calls an appropriate interrupt service routine from the network simulator 105. Each communications controller 204 can be specifically provided to maintain register values and to manage the DMA or interrupt handling operations as needed.

[0026] Communication between the embedded software 103 and the simulated peripherals are replaced by a socket connection to the network simulator system 105. The socket connection can be any two-way communication such as may be provided by TCP/IP.

Alternatively, a serial connection can be provided which maintains a unique communications path between an emulator system 101 and the simulation system 102.

[0027] A stub interface 205 can be provided to manage inter-process communications or communications between the HIL 202 and the HIL Stub 201. Such communications can be made directly to the HIL event handler 206 of the HIL Stub 201. Where a stub interface 205 is used, the stub interface 205 is provided to maintain a main event queue which can be written to by the simulator stub 203 and read by the HIL event handler 206. Alternatively, the main event queue can also be written to by the simulator IP stub 203. The stub interface 205 also maintains a response queue which can be written to by communications controllers 204 and the HIL event handler 206. The response queue of the stub interface 205 can be read by the simulator stub 202.

[0028] The simulator IP stub 203 is provided to encapsulate communications between the embedded software 103 and the simulation system 102. Details of one method of encapsulating communications between the emulator system 101 and the simulation system 102 are described below. The simulator IP stub 203 encapsulates socket communications, transmits and receives socket messages, and delivers socket messages to the HIL event handler 206.

[0029] The HIL event handler 206 is provided to handle incoming data from the simulator IP stub 203 and the stub interface 205, if provided, or the simulator stub 203. The HIL event handler 206 passes responses from read requests back to the embedded software 103 via the stub interface 205.

[0030] Figure 3 shows one embodiment of the network simulator 105. The network simulator 105 communicates with the software emulators 101 and simulates communication

between the embedded software 103 and simulated memory at a register access and at the DMA level. The network simulator 105 also simulates communications between the simulated devices of the simulated network. For example, for read register requests from the embedded software 103, results are sent back to the appropriate device. Read and write actions on registers -- such as a write request to a reset register that causes all other registers in a device to clear their values -- can result in other registers being changed. The network simulator 105 generates interrupts in response to writes to the appropriate bits in all force-interrupt registers. The setting of corresponding interrupt mask registers can prevent the force-interrupt registers from generating interrupts. A read or write operation can cause interactions between nodes in a simulated network, such as a packet being sent, which can evoke register changes, interrupts and other things in other simulated nodes. Nodes represent logical entities which can be provided for by the emulator system 101, the associated emulator stub 301, and the hardware simulator 304. The network simulator 105 runs a simulation of events of all nodes in the network and can thereby provide for connection to several emulator systems 101. In addition, the network simulator can save the current state of a series of register and memory banks and reload that state at a later time.

[0031] By providing additional connections, which can be provided as TCP/IP, the network simulator 105 can communicate with end station simulators 109. The end stations simulators 109 can be simulated or be provided as an actual physical interface. In addition, the network simulator 105 can receive user information and instructions through a remote procedure call from either the scripting interface 107 or the graphical user interface (GUI) 108. The network simulator 105 maintains a topology of the network being simulated as well as state

information for each simulated physical device. The network simulator 105 can inject errors into the network for testing purposes or perform diagnostic operations.

[0032] The network simulator 105 is provided with at least one emulator stub 301 which coordinates communications between the network simulator 105 and each instance of an emulator system 101. Several instances of each emulator system 101 can be handled at once by maintaining a one to one mapping of an emulator stub 301 to each emulator system 101. The emulator stub 301 encapsulates the socket communication, decodes and encodes events to and from the socket messages, transmits and receives messages, and delivers and receives events to and from the network event handler 303.

[0033] Hardware simulators 304 are provided to emulate the software and hardware activities executed on the simulated devices. Each hardware simulator 304 runs a state machine for the hardware of a single device. One to one correspondence of each hardware simulator 304 to an emulated device is provided to ensure consistency in the state of the system. Each hardware simulator 304 also maintains all of the tables and other data that would otherwise be maintained by the real hardware. Such tables can include a packet matching lookup table among other things. In addition, each hardware simulator 304 can be programmed to process information and events as would the real hardware, for example by processing packets. Each instance of hardware simulator 304 is not provided with knowledge of the entire network but is configured as a stand alone component connected to the network according to the network topology. Each hardware simulator 304 receives and sends events to or from the network event handler 303. The events are either events from the embedded software 103, events from an end

station simulator 109, or events from another hardware simulator 304. All of these events are received, processed, and delivered by the network event handler 303.

[0034] The network event handler 303 is provided to perform as the central event processing unit for the network simulator 105. The network event handler 303 receives events from each software emulator 101, each hardware simulator 304, and each end station simulator 109. Events can include any message, control or data that is passed among the hardware simulators 304, the embedded software 103 or an end station simulator 109. The network event handler 303 determines the destination of an event and delivers the event to the appropriate hardware simulator 304, end station simulator 109, or instance of embedded software 103. The network event handler 303 consults with a topology database maintained by a topology manager 302 in order to determine the proper destination of an event. The network event handler 303 can also be provided to log all of the events to a file. Time information related to these events can also be recorded.

[0035] A user event handler 308 can be provided to process commands from the simulator library 106 and to distribute commands to the appropriate simulator component. The user event handler 308 processes commands for populating the topology database as described below. The user event handler 308 also provides for the invocation of the simulated end stations 109, both scripted and real, the injection of errors into the system for testing, as well as the display of diagnostics data from the system. The user event handler 308 provides an active processing thread and passes any asynchronous notifications back from the network simulator 105 to the GUI 108. Alternatively, multiple threads of control can be provided.

[0036] A topology manager 302 is provided to allow a user to create a network topology by providing a set of commands to specify the devices, links, and end stations of the simulated network and is used by the network event handler 303 to map sources to destinations. Sources and destinations are network devices, real or emulated, that can be linked by communications ports through the network simulator 105. The topology manager 302 permits an invocation of the topology and maintains the topology database that can be consulted by the network event handler 303 for the distribution of events. In one embodiment of the simulator 100 according to the invention, there is just a single instance of the topology manager 302, however additional topology managers can be provided. The topology database contains a node table and several link tables. The node table contains information about the logical nodes in the topology where each node is accessed by its LocationID value. Each node table entry contains a link table for providing a mapping from a source (LocationID, Port) to a destination (LocationID, port) for that node in the topology. In addition the topology database maintains properties of the link, such as error conditions. The node table provides a mapping from a LocationID to an EventAcceptor for the LocationID. An EventAcceptor is any device in the network which can accept the event. EventAcceptors include the hardware simulators 304, end station stubs 307 and emulator stubs 301 among other things. The node table also maintains properties of the node, such as error condition, IP address/port number, among others things. The topology manager 302 can optimize the storage of the topology for fast access during simulator operation.

[0037] The end station stub 307 is provided for forwarding messages between an end station simulator 109 and the network event handler 303. One instance of an end station stub 307 is provided for every instance of an end station simulator 109 provided in the network. The

end station stub 307 can communicate with an end station simulator 109 through a communication link which can be provided as TCP/IP.

[0038] An error injector 306 is provided for simulating various error conditions in the simulated network such as dropped packets, erroneous packets, nodes failing, links failing, and card removal. The error injector 306 communicates error requests from the GUI 107 or scripting interface 108 through the user event handler 308 to the network event handler 303 and the topology manager 302.

[0039] A diagnostics module 305 is provided to allow the user to view internal data structures in the simulator system 100. The diagnostics module 305 retrieves data maintained by other portions of the system, such as the hardware simulators 304, and passes data to the GUI 108. The retrieved data can include the contents of hardware tables and registers from the hardware simulator 304 instances, data in the event log maintained by the network event handler 303, or packets exchanged by an end station simulator 109 with an emulator system 101, among other things.

[0040] Another alternative embodiment provides for a hardware generator which permits a user to utilize a GUI 108 to automate the creation of hardware simulators for use in the simulated network. A simulated network or a portion of a network, such as a network element, that has been generated can be saved, adapted and reused for other projects. The hardware generator permits definition of registers with either predefined functionalities or generates frameworks for project specific functionalities. Reusable units may be created by mapping registers into the address space of devices layered within other devices, such as address space of

registers in chips, chips within boards, and boards within backplanes, among other things. Specific functionality can be created according to each layer of abstraction of reusable unit.

[0041] The hardware generator can provide for definition of two broad categories of objects: project objects and hierarchy node objects. Project objects describe the structure of a particular project's nodes, for example, how many shelves, cards, and ports exist in the node. This structure can be used by the simulator GUI 108 in response to a user request to create a new node and for determining how to display the node graphically. Hierarchy node objects describe the memory map of simulated devices. The hierarchy node objects are used by the hardware generator routines for creating simulator code for a simulated device.

[0042] The user can create a network of nodes by providing link interconnections between the simulated nodes and target processes of simulated hardware. Based on a users selection of unit and connections, the hardware generator compiles a graphical representation and produces a set of c++ classes for the simulator and a set of Itkl classes for the simulator GUI. These classes provide frameworks to which special functionality can be added by the user. The simulator can model the structure of network nodes as a set of shelves containing a set of cards. There can be zero to several CPU devices per card. Cards without devices can have their memory-mapped register controlled by another card's CPU device. Once a network definition is complete, a simulation of the network can be invoked.

[0043] In one embodiment, the relationship between project objects of the hardware generator can be provided as shown in Figure 4. A *Project* class can be provided to contain information about the simulator for a project. At least one *ShelfType* class can be provided to

contain a description of one type of shelf. At least one *CardType* class can be provided to contain a description of one type of card. At least one *DeviceDescriptor* class can be provided to contain a description of one of a card's top-level devices. At least one *ObjectDescriptor* class can be provided to contain a description of one of a card's objects (LED or port). At least one *LEDDescriptor* class can be provided to contain a description of how a card's LED object will function. At least one *PortDescriptor* class can be provided to contain a description of how a card's port object will function. At least one *TopLevelDevice* node can be provided as a node in the hierarchy tree where a hardware simulator roots its memory map. The *TopLevelDevice* can be given its own *HardwareSimulator* wrapper class to make it into an EventAcceptor.

[0044] Each project can contain a hierarchy of Tree nodes. Several types of Tree nodes are provided as shown in Figure 5. A *Tree* node can be provided as a basic node type. A *project_node* can be provided as the root of the project's hierarchy. The project node can be provided with the list of *TopLevelDevices* as its children. A *TopLevelDevice* node can be provided as the root of a hardware simulator device. The *TopLevelDevice* can have a list of sub-devices as its children. A *device_node* can be provided which is a generic device node, and can be nested to create a configuration of ASICs within cards, etc. A *bank_node* can be provided which is the base class for banks of memory. The *bank_node* can either be of type memory, register, or a user-defined type. A *memory_bank_node* can be provided which describes a bank of memory using base and top address values. A *register_bank_node* can be provided which describes a bank of registers. The number and location of the registers are specified via the base and top values. Each register contains 16-bits of data and registers are located on even word addresses (i.e., 0x0, 0x2, 0x4, 0x6..). A *register_node* is provided which describes an individual

registers using name, register type, reset value, offset, and mask. The relationship between node types according to one embodiment is shown in Figure 6.

[0045] The following files can be generated by the routines of the hardware generator. A *devices.make* file can be provided which contains the list of the files that need to be compiled for the project. This file can be automatically included in the main makefile for building the projects simulator. A *sim_hardware_launcher.cc* file can be provided to contain the launcher function for inclusion into the topology manager's *CreateDevice* procedure and the *PacketSniffer* function for inclusion into the *EventLogger's* packet dumping procedure. A *sim_{top_level_device}_hardware_simulator.hh* file can be provided which defines the hardware simulator for a top-level device. This file can be created once and can be modified by the simulator developer. A *sim_{top_level_device}_hardware_simulator.cc* file can be provided as the body of the top-level device class. This file is created once and can be modified by the simulator developer. A *sim_{device}_simulator.hh* file can be provided which defines a device's C++ class. This file is only created once and can be modified by the simulator developer. A *sim_{device}_defines.hh* file is provided to contain the definition of constants for the device. A *sim_{device}_simulator.cc* file is provided to contain the implementation of the device's C++ class. This file can be only created once and can be modified by the simulator developer. A *sim_{device}_simulator_init.cc* file is provided to contain the constructor for the device's C++ class. A *{project}.itcl* file can be provided to contain the information necessary by the simulator's iTCL interface (*hw_gui* and *net_gui*) for manipulating the generated simulator. It contains derived iTCL Device classes for each top-level device and a description of the project objects for described how create and display nodes.

[0046] The functions of the hardware generator can include a number of the following processes in creating a simulator. The hardware generator can be provided to create the project's iTCL file, create the top of the *device.make* file, and create the top of the *Hardware Launcher* file. Then for each top-level device, the hardware generator can provide each case statement to the launcher function, create the top-level device's hardware simulator header and implementation file, and create the top-level device's header, implementation, define, and initialization files.

[0047] For each of the top-level device's sub-devices, the hardware generator can create the top-level device's device's header, implementation, define, and initialization files; and add entries to the *devices.make* file. For each of the sub-device's, the hardware generator can be provided to add entries to the parent device's defines file for the sub-device. If the type is a "device", the hardware generator can add an entry to the parent's device's constructor to create the sub-device, and recursively process the sub-device. Otherwise, in one alternative, if the type is a "memory bank", the hardware generator can add an entry to the parent's device's constructor to create the memory bank. In a second alternative, if the type is a "register bank," the hardware generator can add an entry to the parent's device's constructor to create the register bank, and for each of the register bank's registers, the hardware generator can add an entry to the parent device's constructor. Finally, the hardware generator can create a Packet Sniffer that can be used by the simulator for dumping the contents of the packets.

[0048] The GUI 108 can provide a function interface for added project-specific extensions to a basic platform GUI 108. The GUI 108 uses the hardware generator's project objects to determine how to create and display new nodes and endstation simulators 109. Thus,

the hardware generator's project objects act as the recipe to the GUI 108 for how to construct new nodes. The actual instances of the nodes and endstation simulators 109 can be described by another set of iTcl class objects as shown in Figure 7.

[0049] A *Simulator* class can be provided for encapsulation of the simulator-wide functionality, such as invoking, turning logging on and off, etc. A node class can be provided which describes one node in the topology. Each node includes a list of *ShelfObjects*. An *Endstation* can be provided which describes an endstation simulator 109 in the topology. Each *Endstation* can be provided with a name, a hostname, a TCP port number, and executable name, and a *PortObject*. A *Link* class can be provided which describes a connection between two ports in the topology. A *Shelf* class can be provided which contains the description of one of a node's shelves. The description can specify which types of cards occupy each of the shelf's card slots. A *Card* class can be provided which contains a description of one card. The description can contain a pointer to the card's shelf and a list of the card's ports, LEDs and devices. A *Port* class can be provided which describes one port of a card or an endstation simulator's ports. This description can include the port's name, type, directionality, and visibility. An *LED* class can be provided which describes one of a card's LED objects. A *LED* object simulates a Light Emitting Diode by mapping a value stored in a specific address to a set of colors to display. A *Device* class is provided to encapsulate the functionality of a top-level device. It contains the topology manager's ID so that the GUI 108 can identify the device after creation. Each project can derive its own *Device* classes from the *Device* base class. These derived classes can be defined in the {project}.itcl file.

[0050] Although the present invention has been described in connection with specific exemplary embodiments, it should be understood that various changes, substitutions and alterations can be made to the disclosed embodiments without departing from the spirit and scope of the invention as set forth in the appended claims.

reduced content